ArgPROLEG: A Normative Framework for The JUF Theory

Zohreh Shams [1], Marina De Vos [1], Ken Satoh [2]

[1] University of Bath, Dept. of Computer Science, UK
{z.shams, cssmdv}@bath.ac.uk
[2] National Institute of Informatics, Principles of Informatics Res. Devision, Japan
ksatoh@nii.ac.jp

Abstract. In this paper we propose ArgPROLEG, a normative framework for legal reasoning based on PROLEG, an implementation of the the Japanese "theory of presupposed ultimate facts" (JUF). This theory was mainly developed with the purpose of modelling the process of decision making by judges in the court. Not having complete and accurate information about each case, makes uncertainty an unavoidable part of decision making for judges. In the JUF theory each party that puts forward a claim, due to associated *burden of proof* to each claim, it needs to prove it as well. Not being able to provide such a proof for a claim, enables the judges to discard that claim although they might not be certain about the truth. The framework that we offer benefits from the use of argumentation theory as well as normative framework in multi-agent systems, to bring the reasoning closer to the user. The nature of argumentation in dealing with incomplete information on the one hand and being presentable in the form of dialogues on the other hand, has furthered the emergence and popularity of argumentation in modelling legal disputes. In addition, the use of multiple agents allows more flexibility for the behaviour of the parties involved.

Keywords: Legal Reasoning, Normative Framework, Argumentation, Agents

1 Introduction

Legal reasoning is a rich application domain for argumentation in which exchanging dialogues and inferencing are combined [17]. On the other hand, legal reasoning is a rich domain for agent modelling in which agent can model individual parties [14]. In the past two decades, the combination of argumentation and agents technology has provided a great modelling tool for legal disputes, in which multiple parties are involved in a dispute and they each try to prove their claims [3, 16].

In this work, we offer a normative framework for the JUF theory by means of argumentation and multi-agent systems. This allows an easier presentation of this theory, compared to the previous implementation in logic called PROLEG

[22]. The JUF theory is a decision making tool that has already been successfully used in modelling civil litigation [20]. However, having the users - lawyers and judges - of the system in mind, some of the semantics of logic programming does not seem to be fully accessible to the users. We, therefore, have changed the architecture and algorithm of PROLEG in a way that brings the reasoning process closer to the users. For this purpose, we have used the dialectical proof procedure as a reasoning mechanism for parties involved in an argumentation-based dialogue [25]. The advantage of this mechanism is being close to the human reasoning process as well as being representable in form of dispute trees.

This paper is organised as follows. In Section 2 we give an overview of the JUF theory and PROLEG, followed by a brief introduction to argumentation theory and norms. Section 3 provides the main contribution of this paper, which is a normative architecture, called ArgPROLEG. ArgPROLEG is designed for reasoning about JUF theory and in essence, it is an argumentation based implementation of PROLEG. The architecture and algorithm of ArgPROLEG are both included in this section. This section also includes an example of a legal dispute modelled by ArgPROLEG. We then provide a survey of related work in Section 4. Finally we conclude and point out some directions for future work in Section 5.

2 Background

In this section, we provide a brief introduction to JUF, PROLEG and other key concepts used throughout the paper.

2.1 PROLEG: An Implementation of The Ultimate Fact Theory of Japanese Civil Code

PROLEG [20] is a legal reasoning system based on the Japanese theory of presupposed ultimate facts (JUF). This theory is used for interpreting the Japanese civil code. It was mainly developed to assist judges to make decisions under the incomplete and uncertain information they face in the court. This uncertainty is mainly the result of one party asserting a claim, which is unable to prove due to the lack of evidence. In such a situation, the judge cannot deductively decide whether the claim is true or false since the "deductive" civil code is based on the complete information [22].

The JUF theory helps the judge to handle these cases by attaching a *burden of proof* [17] to each claim. The *burden of proof* is assigned to the party that makes the claim and the judge is not responsible for that. Thus, if a party makes a claim that is unable to prove, the judge can discard the claim without trying to assign a certain true or false value to it. This way the judge can evaluate the correctness of a legal claim under a set of incomplete information.

PROLEG was introduced in an attempt to replace an existing translation of the JUF theory into logic programming [22]. The reason of this shift was the unfamiliarity of the users, namely judges and lawyers, with logic programming and

negation as failure [5] in particular. According to *negation as failure*, if a claim is unknown or not known to be true, it is considered to be false. By definition, *negation as failure* makes a perfect choice for a mathematical formalisation of the JUF theory in which failing to provide a proof for a claim results in discarding the claim. However, the fact of not being conceptually accessible for the users, led to a new implementation of JUF called PROLEG.

Instead of *negation as failure*, PROLEG uses the Professor Ito's explanation of JUF which is based on the openness of the ultimate facts [20]. In openness theory, facts are divided into two categories; those that result in a conclusion and those that represent an exceptional situation. The latter category are open to challenge meaning they do not have a certain truth value and are therefore undecided form the judge point of view. The *burden of proof* of these facts is on the party claiming them. Judges are therefore able to make decisions based on known facts and exceptions that are explicitly proven by one of the parties.

PROLEG consists of a rulebase and a factbase. The former stores the rules and the exceptions while the later stores the performed actions of both parties as well as the judge's judgement about their truth value. Equations (1), (2) and (3) are examples of a rule, an exception and a fact in PROLEG, respectively.

$$deliver\_good(X, Y, Good) \quad <= \quad purchase\_contract(X, Y, Good, Price) \quad (1)$$

$$exception(deliver\_things(X, Y, Good, Price), \\ claim\_of\_simultaneous\_performance(Y, X, Price)) \quad (2)$$

$$allege(claim\_of\_simultaneous\_performance, plaintiff) \quad (3)$$

Rule (1) states that party X can expect party Y to deliver a $Good$ if there is a purchasing contract between them including the agreed $Price$ and $Good$. However there could be an exception to this expectation, which is defined in rule (2). The exception is as follows: if there is a contract between two parties, one may refuse to perform her/his obligation until the other party performs her/his obligation. Moreover, equation (3) shows a performed action by the plaintiff party, which is claiming an exception to $deliver\_things(X, Y, Good, Price)$ by, $claim\_of\_simultaneous\_performance$.

According to the claims and proofs that two parties - Plaintiff and Defendant - assert, PROLEG produces a trace of derivation in the form of an dialogue between them. The plaintiff tries to prove a claim while the defendant tries to find an exception for that claim. If the exception is proven successfully, the plaintiff has to find another exception for the former exception and so on.

2.2 Argumentation

Argumentation theory was initially studied in philosophy and law, and during the past two decades it has been extensively researched in distributed systems. Argumentation Frameworks (AF) have particularly gained a popularity in multi-agent systems as an aid for the agents' reasoning and decision making process.

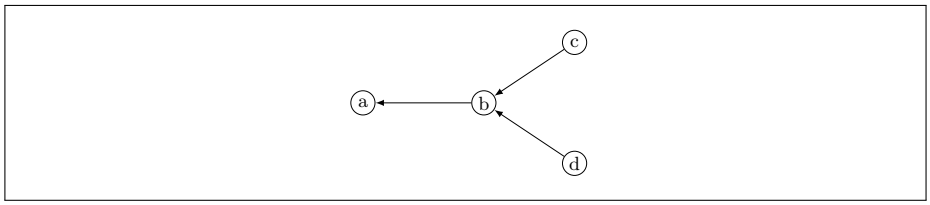

Fig. 1. A Graphical Representation of AF

The first AF was introduced by Dung [7] and it is known as Dung's Argumentation Framework (DAF)[1]. According to DAF, an AF is a pair $AF = \langle Ar, R \rangle$ where $R \subseteq Ar \times Ar$. Ar is a set of arguments and R is a set of attack relations between arguments. We assume a attacks b if $(a, b) \in R$. Figure 1 displays an AF with four arguments and three attack relations between them. Nodes represent the arguments, while edges represent the attack relations among them.

$$AF = \langle \{a, b, c, d\}, \{(b, a), (c, b), (d, b)\} \rangle$$

The evaluation of arguments in an AF depends on the argumentation semantic of choice. The purpose of argumentation semantics is to determine a set of justified and coherent arguments based on the arguments' interactions. If two arguments attack each other then an entity - which could be an agent for example - cannot believe in both of them at the same time. Therefore, the role of argumentation semantics is to examine the acceptability of a set of arguments.

The most basic argumentation semantic is the conflict-free semantics [7] in which none of the arguments attack each other. This is the minimum criteria for a set of arguments to be considered as coherent. The rest of argumentation semantics (e.g. complete extension, preferred extension, stable extension and etc.) are a version of conflict-free semantic that satisfy some form of optimality [6]. As an example, the conflict-free extensions of Figure 1 are provided below:

$$C\_F : \{\{\} \{a\} \{b\} \{c\} \{d\} \{a, c\} \{a, d\} \{c, d\} \{a, c, d\}\}$$

One of the reasons of developing argumentation theory in multi-agent society is being able to present interactions in the form of dialogues, specially among participants with potentially conflicting viewpoints. Dung [8] states argumentation as a form of reasoning for dispute resolution in which two parties, proponent and opponent, engage in a discussion as a form of proof for their claims. In fact, dialectical proof procedure can be viewed as a reasoning mechanism for parties involved in an argumentation-based dialogue [25]. In such a dialogue, the proponent puts forward an argument with the purpose of proving it. However, the opponent tries to attack this claim. The dispute goes on by the proponent and

[1] DAF can also be referred to as an Abstract AF because it abstracts away the internal structure of arguments and instead, it merely focuses on attack relations among arguments.

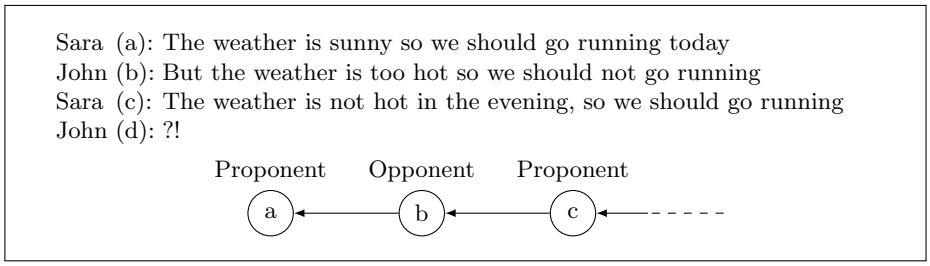

Fig. 2. Dialectical Proof Procedure

opponent alternating in attacking each other's previous arguments until one of them runs out of arguments. The winner of the dispute is the party who speaks last. Therefore, the original claim by proponent is proved if the dialectical proof procedure ends with an argument by proponent. Figure 2 shows an example of this nature, in which the proponent claim is accepted.

2.3 Norms

Norms are defined as social rules which control the agent society by regulating agents' behaviour and following them benefits multi-agent systems as a whole [26]. They help multi-agent systems to cope with the heterogeneity, the autonomy and the diversity of interests among agents [27]. Therefore, a normative framework for multi-agent systems, comprises a set of normative agents whose behaviour is governed by norms [27]. If these norms are legal norms, then we have a legal normative framework which is the focus of this work. Free Online Encyclopaedia defines legal norms as "mandatory rule of social behaviour established by the state". As this definition suggests, legal norms are normally imposed to the society through an external entity such as the state. We have thus, adopted the same concept and modelled the legal norms external to the agents.

We define each norm as a rule of form (4) consisting of literals Li.

$$L_0 \leftarrow L_1 \wedge \cdots \wedge L_m \qquad m \geq 0 \tag{4}$$

The left hand side of the arrow L_0 is called head or conclusion of the rule and the right hand side $L_1 \wedge \cdots \wedge L_m$ is called body or premises of the rule. L_0 holds if $L_1, L_2, \cdots$, and L_m are all true. Take for example the norm:

$$payfine(AgX, Y) \leftarrow delay(AgX, Y) \wedge reserved(Y) \tag{5}$$

This norm can be read as: Agent AgX has to pay fine if it delays returning book Y to the library and the book is reserved by someone else.

Since we aim to use norms in a legal reasoning context, as it has been used in the JUF approach presented by PROLOG, we require a second type of norm called an exception norm.

$$Exception(Q, P) \tag{6}$$

Equation 6 states, that there is an exception, namely P for Q which is the head of another norm.

Exception norms substitute the facts representing exceptional situations in PROLEG (see Section 2.1).

$$Exception(payfine(AgX, Y), available(Y, Y^{'})) \tag{7}$$

For example, assuming $Y^{'}$ is a second version of book Y, the above norm reads as: AgX does not have to pay fine if another version of book Y, $Y^{'}$ is available.

3 ArgPROLEG: A Normative Framework for Legal Reasoning

The JUF theory was first implemented in logic programming followed by an implementation in prolog called PROLEG. The main advantage of PROLEG over the original system is its accessibility to lawyer and judges. In this section we propose a normative framework to model the JUF theory which is even closer to the natural human reasoning process, since it benefits from multi-agent systems and argumentation theory to represent a legal dispute between two parties, namely plaintiff and defendant.

Arguing is one of the human skills that we learn from early ages. Naturally, the argumentation process between two humans starts with one of them raising an issue which is subject to disagreement of the opposite party. The rest of process is followed by exchanging further arguments with the purpose of reaching an agreement. The agreement could be mutual or could be the result of one party not being able to reject the other party's argument. Similarly, we have tried to reflect the human reasoning process in ArgProleg in a way that even a non expert user can instinctively relate to it. In what follows, We first introduce the overall architecture of ArgPROLEG followed by its algorithm.

3.1 The Framework Architecture

We suggest an architecture (see Figure 3) in which the two parties in a legal dispute, plaintiff and defendant, are presented by two agents A and B, respectively. The arbitrator plays the role of the judge in the court and the set of norms models the law book. The arbitrator receives the claims and evidences of each parties and judges them by referring to the set of norms.

Please note that the connection between two agents happens through the arbitrator (See Figure 3) and there is no direct connection between agents. A legal case in the court normally commences with an agent, namely plaintiff raising an issue against another agent, namely defendant. It is then the judge's

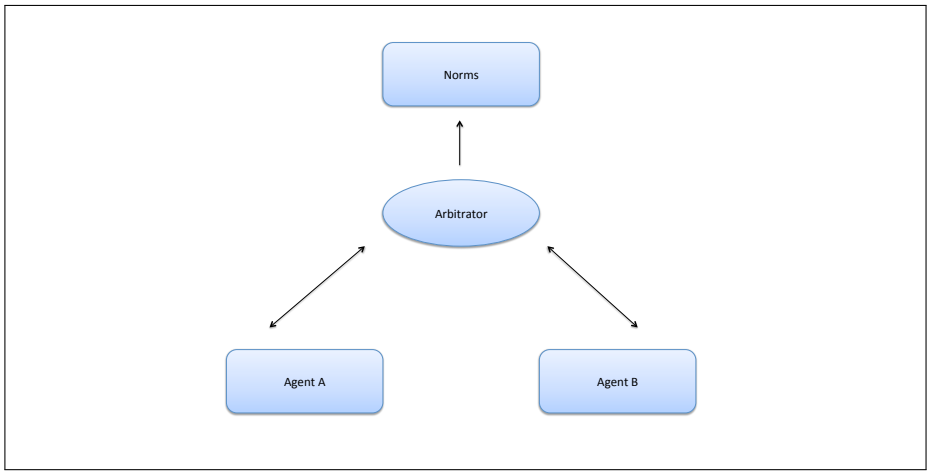

Fig. 3. The Framework Architecture

responsibility to investigate the raised case and ask for defendant's testimony. According to both the original claim by plaintiff and the provided response by defendant, the judge decides whether the dispute is over in favour of one of the parties. If the status of the dispute is still unclear to the judge, it will be more argumentation back and forth between two parties through the arbitrator. Below is a narrative on how the communication works between the various parties:

- The session starts by agent A submitting a claim to the arbitrator.
- The arbitrator checks the set of norms to find out how agent A should support this claim. In other words, what are the requirements of this claim from the legal viewpoint.
- The arbitrator passes the requirements to agent A.
- If agent A fails in providing the requirements, the claim is rejected.
- If it succeeds then, the arbitrator contacts the set of norms to see if there are any exceptions for this claim. If not, the claim is accepted.
- Otherwise the arbitrator passes the exceptions to agent B to see if it can provide any of them [2].
- If agent B has any of those exceptions, it will then pass it to the arbitrator.
- Subsequently, the arbitrator tries to find out how this exception can be supported from the law viewpoint by referring to the set of norms.
- The arbitrator informs agent B about the required support.
- If agent B cannot provide the necessary support for any of the exceptions, agent A's claim is accepted.

[2] We assume that a party can use all the exceptions available exhaustively, one-by-one, to make a successful counter attack. Thus, if the party cannot provide the required support for the first exception, it has the opportunity to try the second exception and so on.

– But if agent B can prove at least one of the exceptions, the arbitrator tries to find out what are the exceptions for that by checking the set of norms.
– If there is any they will be passed to agent A and the same procedure will be repeated.
– This procedure is repeated until either an exception to the original claim cannot be ignored (the plaintiff cannot counteract) or all exceptions to the original claim turn out to be unsupported by the defendant.

3.2 The ArgPROLEG Algorithm

The ArgPROLEG algorithm (Figure 4) consists of six functions: **Main(C)**, **prove(S,P)**, **provide-evidence(M)**, **claim(A,B)**, **reverse(X)** and **except(F,Q)**. The task(s) that each function fulfils is explained below.

The **Main(C)** function returns the output of **prove(S,P)** function for the plaintiff's original claim, **C**. The **prove(S,P)** function is used to prove a claim or the support of an exception by either parties. If a party **P** puts a claim or an exception forward, the arbitrator will check the the set of norms to see how the claim or exception can be proven. The arbitrator then asks the agent to provide the proof of the claim by showing evidence. If the agent can provide the necessary evidence by means of **provide-evidence(M)** function, the evidence is passed to the **claim(A,B)** function to see if there is any indirect attack to the original claim **C**. By indirect attack, we mean an exception to any part of the evidence of **C**. Therefore, the proof is successful if evidence is provided and all claims against it are rejected. The **provide-evidence(M)** function is a function that is used by each single agent collecting all the rules that have **M** as their head. It then recursively, traces back each rule to find all its atoms. The output of this function is a set of sets. Each set provides a possible way to proof the claim. For example in the case provided below, the agent has to provide $\{p1, p2, p3, p4, p5, p6, M\}$ or $\{q1, M\}$.

$R1 : M \Leftarrow p1, p2$	$R4 : p2 \Leftarrow p4, p5$	$R7 : p3 \Leftarrow$
$R2 : M \Leftarrow q1$	$R5 : p4 \Leftarrow p6$	$R8 : p5 \Leftarrow$
$R3 : p1 \Leftarrow p3$	$R6 : q1 \Leftarrow$	$R9 : p6 \Leftarrow$

Function **claim(A,B)** takes the responsibility of the rest of the dispute after the first claim by plaintiff is proven to be true. This function then gives chances to the defendant and the plaintiff to attack each other's last announcement. If any of the exceptions against an argument remains unattacked by the other party, that means the dispute is over and the winner is the claimer of this argument. The output of this function is **true** if **A** who made the first claim/argument is the last who speaks. Otherwise the output is **false**. The **reverse(X)** function takes one of the parties, either the plaintiff or the defendant as input and returns the opposite party as output. This function will be called in **claim(A,B)** function, when the parties have to take turn in attacking each other.

```
Plaintiff-Arbitrator: Main(C)
begin
    return(prove(C, Plaintiff))
end

prove(S,P)
begin
    Arbitrator-P: Provide evidence for S
    V = provide − evidence(s)
    if V = Ø then return(false);
    for every v ∈ V
        begin
            proven = true
            for every v_i ∈ v
                begin
                    if claim(v_i, P)
                    proven = false
                    break
                end
            if proven = true
            return(true)
        end
    return(false)
end

provide-evidence(M)
begin
    Result = {}
    R_u = {M ⇐ D ∈ R}
    if R_u = Ø then return(Ø)
    for every R_i ∈ R_u
        begin
            if D = Ø then add {} to all sets in Result
            else if for all D_i ∈ D
                begin
                    add provide − evidence(D_i) to all sets in Result
                end
                add {D} to all sets in Result
        end
        return(Result)
end

claim(A,B)
begin
    e = except(A, reverse(B))
    if e = Ø then return(true)
    else for all e_i ∈ e
        begin
            result = claim(e_i, reverse(B))
            if result = true then return(false)
        end
    return(true)
end

reverse(X)
begin
    if X = Plaintiff then return(Defendant);
    else return(Plaintiff);
end

except(F,Q)
begin
    Arbitrator-Norms: collect all the exceptions for F: exception(F, E_i) in E
    if E = Ø then return(Ø)
    provenE = Ø
    else for every E_i ∈ E
        begin
            Arbitrator-Q: evidence(E_i)
            if Q can provide the evidence
                then Arbitrator-Q: prove(E_i, Q)
                    if prove(Q, E_i) = true
                        then provenE = provenE ∪ E_i
            else return(Ø)
        end
    return(provenE)
end
```

Fig. 4. The ArgPROLEG Algorithm

The **except(F,Q)** function tries to find the exceptions for a certain claim or exception, **F**. If **F** is provided by one party, the opposite party **Q** needs to show evidence and consequently prove the exceptions for **F**. Thus, the arbitrator checks the set of norms to see whether there is any exceptions for **F**. In case of existence, the exceptions will be passed to **Q**. This party has to firstly show an evidence of such an exception and secondly prove it by calling **prove(S,P)** function. If it fails either of them, then the exception is rejected. The output of this function is either Ø, which means there is no exception or not any proven one for **F**; or it is set **provenE** which is a set of proven exceptions for **F**.

3.3 Contract Scenario

In this scenario, we aim to model a legal dispute between two parties by means of the architecture and the algorithm we introduced in Sections 3.1 and 3.2. Imagine a situation in which a lessor wants to cancel his property contract with the lessee. She claims that the lessee has subleased the property to somebody else and therefore, she wants to end the contract. Both the lessor and the lessee agree that there was a contract between them in first place and subsequently the property was handover to the lessee. The lessee also admits her contract of sublease with a third person which was followed by handing over some parts of the property to the sublessee. The lesser believes that the sublease has used the property to make profit, thus she makes the announcement of cancelling the contract. However, the lessee believes that she already informed the lessor and she has approved of the sublease before she made the announcement of cancelling the contract. Moreover, the period of subleasing was so short that does not count as abuse of confidence of the owner. However, the owner considers the case as abuse of confidence since she has received some complaints from the neighbours regarding the noise during the subleasing period. Figure 5 displays the formalisation of this case based on the ArgPROLEG architecture.

Figure 6 illustrates the graphical representation of Contract Scenario based on the ArgProleg algorithm. The plaintiff claims that she wants to cancel the contract. The arbitrator then checks the set of norms to find out the support for this claim. N1 provides this information which will be passed to the plaintiff. Plaintiff is able to provide the required support. Thus the first argument (a) appears. The arbitrator checks the set of norms to see if there is any exceptions for this claim. Exceptions 1 and 2 provide two options for the defendant to make an attack against the plaintiff's claim. The options obtained from the exceptions are $b : get\_approval\_of\_sublease$ and $c : nonabuse\_of\_confidence$. N2 and N3 contains the necessary supports for each of the exceptions, respectively. The attack (b) and (c) to (a) remains as a potential attack unless the defendant can provide the requested support for them. Defendant can only provide this support in case of argument (c). Therefore, the defendant attacks argument (a) by argument (c). Now, based on the algorithm, the arbitrator checks the set of norms to find an exception to this exception. This is going to make a potential case for the plaintiff to perform an attack to the defendant. There is one exception available, namely Exception 3, $abuse\_of\_confidence$. N4 states

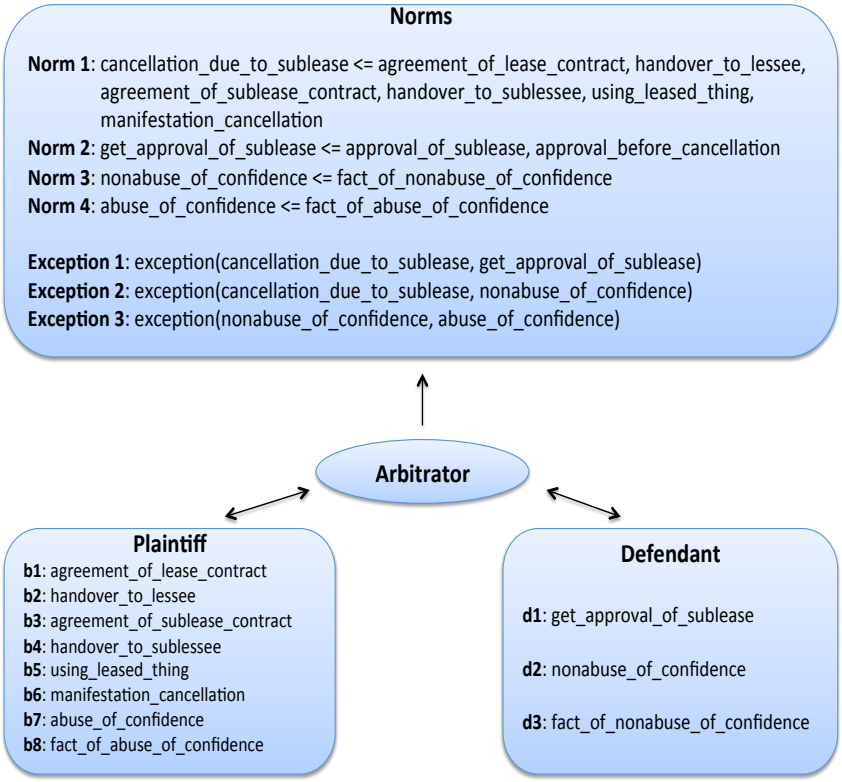

Fig. 5. Contract Scenario

the requirement for this argument, which is $fact\_of\_abuse\_of\_confidence$. The plaintiff successfully supports this argument which results in an attack from argument (d) to argument (c). The arbitrator looks for another exception to this later exception. Since such an exception is not available the dispute is over.

The last graph in Figure 6 shows the final argumentation framework for this dispute. Going back to Section 2.2, in a dialectical proof procedure, the party who makes the last utterance is the winner, which similarly makes the plaintiff the winner of this case.

4 Related Work

The closest work to ours is PROLEG [20] which is an implementation of JUF theory by means of the *burden of proof*. ArgPROLEG has fulfilled two future plans of PROLEG discussed and listed in [20]. These two features are, bringing the knowledge representation closer to the natural human reasoning (see section 3) and also including a diagrammatic representation of reasoning in the

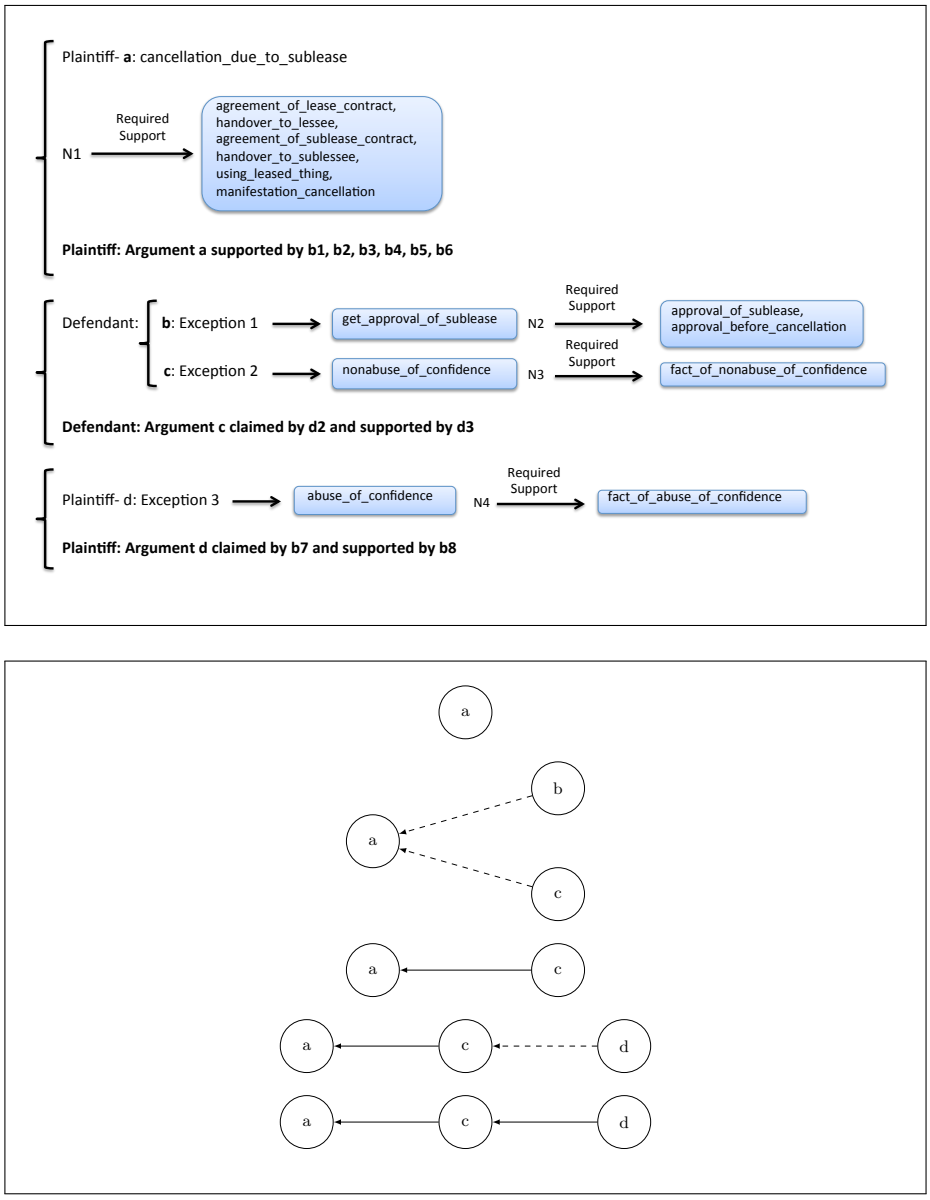

Fig. 6. Contract Scenario Argumentation Framework

JUF theory. Using argumentation in designing ArgPROLEG has served both these purposes.

Apart from PROLEG and ArgPROLEG, another translation of the JUF theory is also available in logic programming [22]. In contrast to PROLEG and

ArgPROLEG, this version uses *negation as failure* instead of the *burden of proof*. *Negation as failure* is a non-monotonic form of negation that enables logic programming to formulate problems of non-monotonic reasoning. Kakas [15] has already used *negation as failure* for default reasoning. The idea of using negative literals as abductive hypotheses has also been pointed out by Eshghi and Kowalski [9]. However, among *burden of proof* and *negation as failure*, the concept of the former seems to be easier to grasp for lawyers and judges.

In terms of formalisation of the *burden of proof*, other works exist [13, 18, 23, 28]. Gordon et al. offer an argumentation-based system, called Carneades [13], which implements the *burden of proof* as well the *burden of persuasion*. The main difference of this approach to ours is, that the *burden of proof* for a premise can be assigned to a different party rather than the one who has uttered the claim. The plaintiff has the *burden of production* for the facts of its claim, whereas the defendant has the *burden of production* for exceptions. The same applies to the *burden of persuasion*.

Another example of logic programming being used for expressing and applying legislation is [24]. This work however, focusses on specific legal cases related to British Nationality Act. They describe how complicated regulations such as British Nationality Act can be translated into simple form of logic so that the consequences of each act can easily be determined.

5 Conclusion and Future Work

In this paper we introduced ArgPROLEG, a normative framework for legal reasoning, that uses dialectical proof procedure to support legal parties in resolving their conflicts. ArgPROLEG offers an alternative approach to PROLEG [20]. We believe that ArgPROLEG is closer to human reasoning compared with PROLEG. Additionally, it is able to offer a diagrammatic representation of the plaintiff's and the defendant's reasoning, which enhances the ability of non-expert users to follow the procedure as it unfolds.

For the future, we would like to extend our framework to be able to cope with more than two parties. In real cases, a dispute can involve multiple parties, which all need to be able to bring forward their arguments. If there are more than two agents involved, but we are still able to divide them into two main opposing parties, the argumentation graph keeps its linear shape. However, in each step, there is more than one agent that can put forward an argument. For instance, if it is the defendant party's turn to put forward an argument and the defendant party includes more than one agent, any of them can make the argument. Although from the argumentation graph viewpoint, the dialectical structure does not change, there are some other issues that need to be taken into consideration. The most important issue is the consistency of knowledge bases of different agents belonging to the same party. At the moment we assume each agent's knowledge base is self-consistent, which results in consistency of claims put forward by the agent. However, if a party consists of more than one agent, defending the same viewpoint, their claims have to be consistent

too. One possible way of achieving consistency is to merge agents' knowledge bases and resolve possible conflicts to prevent any inconsistent arguments. One example of such an approach is discussed in [10]. Another issue is the method of constructing an argument. Arguments can be put forward by various individual agents belonging to the same party, as well as by a combination of agents. The process of construction of an argument in such cases, results from the reasoning of multiple agents. On the other hand, if there are more than two agents and we cannot simply divide them into two main opposing parties, the argumentation graph would not be linear any more. As a result the argumentation graph can take any shape and the winner of the dispute is not necessarily the party who speaks last. In addition, applying different argumentation semantics, as discussed in section 2.2, would have a different outcome, whereas in linear graphs all the argumentation semantics coincide.

For the implementation, we consider an architecture similar to the Governor approach presented in [1]. Balke et al. use an institution to collect the norms and the normative results of an agent's actions. To make this information accessible via queries, the authors introduce the Governor, an agent that acts as a relay between the norms and their (possible) consequences and the agent's query. In our case, the arbitrator would take the role of the Governor. Apart from simply relaying queries to the institution/norms, the arbitrator will actively retrieve information to pass on to the agents, e.g. the exception to the claim. [1] uses the Jason BDI architecture [4] for setting up the multi-agent system and InstAL based on answer set programming [2, 12] for the institution/norms. The use of a BDI architecture [19] has the added advantage of being able to model agent reasoning in more detail. Currently, our agents' mental model contains only beliefs or a knowledge base. In a BDI architecture, we could express the goals and intentions of the agents more effectively and take them into account when they put forward their claims. In addition, with an institution rather than a set of norms, we would be able to keep track of normative states and allow agents to reason about conflicts that appear after a period of time. Having more expressive agents, gives us the chance to investigate different strategies for agents to deal with norm compliance as well. Agents can check the reward and punishment associated with adhering to a norm, or violating a norm, and decide whether the gain from breaking a norm is worth the sanction. In such cases, the agent has to decide between the importance of individual goals hindered by normative goals compared to individual goals hindered by punishment [27].

For the dialectical proof procedure, we also consider an implementation using answer set programming (ASP). Dung's argumentation framework and semantics have already been implemented in answer set programming [11], giving us a good indication that this approach is worth considering.

Finally, this paper only investigates the use of argumentation for conflict resolution in a legal domain. In particular, it is aimed at mimicking court procedure. However, it has been proven [21], that the PROLEG inference structure can be used for general *rule - exception* patterns (see section 2.3). This argument applies to ArgPROLEG as well since it borrows PROLEG inference structure.

References

1. Tina Balke, Marina De Vos, Julian Padget, and Dimitris Traskas. On-line reasoning for institutionally-situated bdi agents. In Yolum, Tumer, Stone, and Sonenberg, editors, *10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 1109–1110. IFAAMAS, May 2011.
2. Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, New York, NY, USA, 2003.
3. Trevor Bench-Capon, Henry Prakken, and Giovanni Sartor. *Argumentation in Artificial Intelligence*, chapter Argumentation in Legal Reasoning, pages 363–382. Springer, 2009.
4. Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.
5. Keith L. Clark. Negation as failure. In Jack Minker, editor, *Logic and Data Bases*, volume 1, pages 293–322. Plenum Press, New York, London, 1978.
6. Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Prudent semantics for argumentation frameworks. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 568–572. IEEE Computer Society, 2005.
7. Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.
8. Phan Minh Dung and Phan Minh Thang. A unified framework for representation and development of dialectical proof procedures in argumentation. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence(IJCAI)*, pages 746–751, 2009.
9. Kave Eshghi and Robert A. Kowalski. Abduction compared with negation by failure. In *ICLP*, pages 234–254, 1989.
10. Xiuyi Fan, Francesca Toni, and Adil Hussain. Two-agent conflict resolution with assumption-based argumentation. In *Computational Models of ArgumentComputational Models of Argument (COMMA)*, pages 231–242, 2010.
11. Sarah Alice Gaggl. Solving argumentation frameworks using answer set programming. Master's thesis, Technische Universitt Wien, 2009.
12. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.
13. Thomas F. Gordon, Henry Prakken, and Douglas Walton. The carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10-15):875–896, July 2007.
14. Thomas F. Gordon and Douglas Walton. Legal reasoning with argumentation schemes. In *International Conference on Artificial Intelligence and Law (ICAIL)*, pages 137–146. ACM, 2009.
15. Antonis C. Kakas. Default reasoning via negation as failure. In Gerhard Lakemeyer and Bernhard Nebel, editors, *ECAI Workshop on Knowledge Representation and Reasoning*, volume 810 of *Lecture Notes in Computer Science*, pages 160–178. Springer, 1992.
16. Henry Prakken. Formalising ordinary legal disputes: a case study. *Artificial Intelligence and Law*, 16(4):333–359, 2008.
17. Henry Prakken and Giovanni Sartor. Formalising arguments about the burden of persuasion. In *Proceedings of the 11th international conference on Artificial intelligence and law*, ICAIL '07, pages 97–106, New York, NY, USA, 2007. ACM.

18. Henry Prakken and Giovanni Sartor. More on presumptions and burdens of proof. In Enrico Francesconi, Giovanni Sartor, and Daniela Tiscornia, editors, *JURIX*, volume 189 of *Frontiers in Artificial Intelligence and Applications*, pages 176–185. IOS Press, 2008.
19. Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In *Proceeding of the first international conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.
20. K. Satoh, K. Asai, T. Kogawa, M. Kubota, M. Nakamura, Y. Nishigai, K. Shirakawa, and C. Takano. Proleg: An implementation of the presupposed ultimate fact theory of japanese civil code by prolog technology. In Takashi Onoda, Daisuke Bekki, and Eric McCready, editors, *JSAI-isAI Workshops*, volume 6797 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2012.
21. K. Satoh, T. Kogawa, N. Okada, K. Omori, S. Omura, and K. Tsuchiya. On generality of proleg knowledge representation. In *Proceedings of the 6th International Workshop on Juris-informatics (JURISIN 2012)*, pages 115 – 128, Miyazaki, Japan, 2012.
22. K. Satoh, M. Kubota, Y. Nishigai, and C. Takano. Translating the japanese presupposed ultimate fact theory into logic programming. In *Proceedings of the 2009 conference on Legal Knowledge and Information Systems: JURIX 2009*, pages 162–171, Amsterdam, The Netherlands, The Netherlands, 2009. IOS Press.
23. Ken Satoh. Logic programming and burden of proof in legal reasoning. *New Generation Comput.*, 30(4):297–326, 2012.
24. M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Commun. ACM*, 29(5):370–386, May 1986.
25. Phan Minh Thang, Phan Minh Dung, and Nguyen Duy Hung. Towards a common framework for dialectical proof procedures in abstract argumentation. *Jornal of Logic and Computation*, 19(6):1071–1109, 2009.
26. Fabiola López y López and Michael Luck. A model of normative multi-agent systems and dynamic relationships. In Gabriela Lindemann, Daniel Moldt, and Mario Paolucci, editors, *Regulated Agent-Based Social Systems (RASTA)*, volume 2934 of *Lecture Notes in Computer Science*, pages 259–280. Springer, 2002.
27. Fabiola López y López, Michael Luck, and Mark d'Inverno. A normative framework for agent-based systems. In *Normative Multi-Agent Systems (NORMAS)*, pages 24–35, 2005.
28. Hajime Yoshino. On the logical foundations of compound predicate formulae for legal knowledge representation. *Artificial Intelligence Law*, 5(1-2):77–96, 1997.

Responses to Reviewers' Comments

REVIEW 1
OVERALL EVALUATION: 2 (accept)

The revised version is eligible for LNAI publication. Some points are made clear, and some related references are included in the new version. It is a good possibility to publish this paper in LNAI for JURISIN 2013. The third paragraphs in section 5 seems not be in a good form. Please take care of the typesetting of the paper before publishing.
Section 5 has been proofread and some sentences have been rephrased.

REVIEW 2
OVERALL EVALUATION: 2 (accept)

The paper has been properly revised according to the reviewer's comments.